



XAPP662 (v2.4) May 26, 2004

In-Circuit Partial Reconfiguration of RocketIO Attributes

Author: Vince Eck, Punit Kalra, Rick LeBlanc, and Jim McManus

Summary

This application note describes in-circuit partial reconfiguration of RocketIO™ transceiver attributes using the Virtex-II Pro™ internal configuration access port (ICAP). The solution uses a Virtex-II Pro device with an IBM PowerPC™ 405 (PPC405) processor to perform a partial reconfiguration of the pre-emphasis (TX_PREEMPHASIS) and differential swing control (TX_DIFF_CTRL) attributes for RocketIO multi-gigabit transceivers (MGTs). These attributes must be modified to optimize the MGT signal transmission prior to and after a system has been deployed in the field. This solution is also ideal for characterization, calibration, and system testing.

The hardware and software elements of this solution can be easily integrated into any Virtex-II Pro design already utilizing the PLB bus structure. This application note uses the reference design accompanying [XAPP661](#), which implements peripherals with a Xilinx Intellectual Property Interface (IPIF) to the PLB bus. A PLB ICAP module is instantiated and controlled through software running on the PPC405 processor. Design modules to facilitate bit-error rate tests (BERT) and pseudo-random binary sequence (PRBS) diagnostics are also included. A terminal interface over a serial port connection allows users to set and modify MGT attributes via a command line.

Application note [XAPP660](#) documents the partial reconfiguration of MGT attributes using a register interface. This black box approach provides a thin IP software based solution for designers not originally intending to use the PPC405 processor in their designs. This can be a preferred solution for updating the MGT attributes once a slot ID has been sampled, after power has been applied to a board. The application developer can later expand the capabilities of the PPC405 in the design through the IBM CoreConnect™ (PLB, OPB) modular features described in this application note.

Introduction

Virtex-II Pro devices contain between four and 20 RocketIO MGTs for the creation of high-speed serial links between devices (up to 3.125 Gb/s per channel). Variations in the system environment can impact the reception of the high-speed serial signals. In these situations it can be beneficial to modify the pre-emphasis or differential swing control levels for one or all of the MGTs on a Virtex-II Pro device while leaving the rest of the FPGA design unchanged. For example, when using the MGTs to create high-speed serial links across a backplane, the distance the signals must travel can change significantly depending on the slot position of the board. Adjusting the pre-emphasis level to compensate for the change in distance allows for the highest quality signal transmission at the intended baud rate.

The MGTs have four levels of pre-emphasis and five levels of differential swing control selectable by RocketIO primitive attributes. These attributes are initially set in the bitstream by BitGen, but can be modified by the use of ICAP after the device is initially configured. These attributes can be modified to optimize the transmission of the high-speed serial signals. For more details on these attributes or the MGTs in general, please refer to the [RocketIO Transceiver User Guide](#).

© 2003-2004 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

This application note demonstrates a partial reconfiguration controller using software coded in C running on the PPC405 processor to modify the MGT pre-emphasis (TX_PREEMPHASIS) and differential swing control (TX_DIFF_CTRL) attributes. After power-up and initial configuration with a base FPGA design, the user commands the PPC405 processor to modify the MGT pre-emphasis and differential swing control attributes. The PPC405 modifies the MGT attributes by first reading the contents of the specified MGT configuration frame through ICAP. The desired MGT attribute change is then performed on the recently read and stored frame. Lastly, the modified frame (held in block RAM or external RAM) is written out to the MGT configuration memory through the ICAP. The configuration frame data is modified and written to the ICAP. This flow is described as the Read-Modify-Write (RMW) of a configuration frame. No partial bitstreams or external devices are required to implement this solution. This application note describes the ICAP IPIF and the software flow necessary to perform the in-circuit update of the MGT frame.

Hardware Implementation

Overview

Flexible system upgrade solutions can be added to a Virtex-II Pro design by augmenting a base system consisting of the PPC405, block RAM or external RAM, and the CoreConnect bus structure with the ICAP module. The Virtex-II Pro RocketIO MGTs support multiple serial I/O standards. These standards can be implemented by tailoring the attributes of the RocketIO transceivers driven by a variety of custom hardware IP modules. The ICAP module and accompanying software provides a common solution for modifying the MGT attributes as well as a pathway for system upgrades.

The RocketIO transceiver BERT reference design in XAPP661, shown in [Figure 1, page 3](#), provides a specific example of a system that includes an ICAP module and a custom hardware module. XAPP662 presents the underlying in-circuit partial reconfiguration technology of the ICAP module instantiation and the associated adaptive software.

The PPC405 processor uses the ICAP to perform in-circuit updates of configuration frames. The design in XAPP661 demonstrates a system using PLB and DCR devices. A PLB implementation for the ICAP IPIF is integrated for convenience purposes since all other devices in the design use the PLB bus. The XAPP661 design consists of four PLB slave devices. A block RAM (BRAM) controller connects 32 KB of BRAM to the bus serving as the data and instruction memory of the processor as well as scratch pad storage for configuration frame data. A UART module, a two-channel XBERT module, and an ICAP module each bonded with an IPIF provide standardized connections to the PLB bus. The processor has two PLB master connections, one for instruction cache and one for data cache. The bus error address and status register of the PLB arbiter is the only DCR device in the reference design. XAPP661 contains a more detailed description of the BERT design. Many embedded solutions are integrated in a modular fashion on boards and in a backplanes. This same embedded development approach, using the IBM CoreConnect and IPIF, is now available internal to the FPGA.

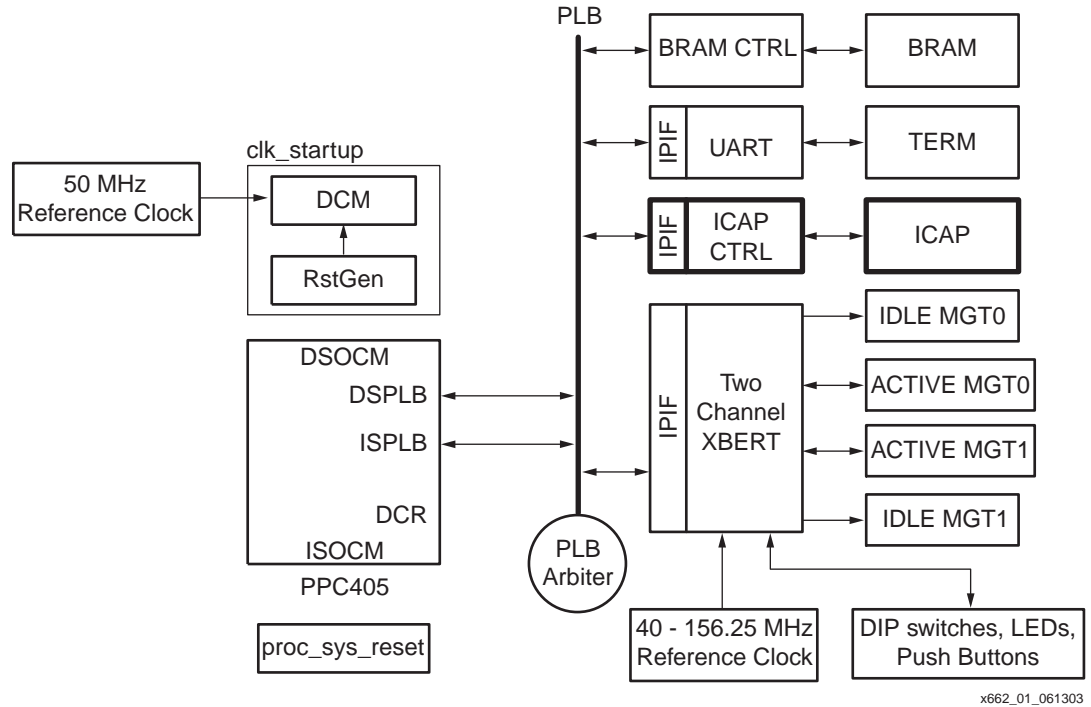


Figure 1: High-Level Hardware View of the RocketIO Transceiver BERT Reference Design with ICAP Module Instantiation

ICAP Module

The ICAP module is the fundamental module used to perform in-circuit reconfiguration in Virtex-II Pro devices. The ICAP block is located in the lower right hand corner of the FPGA. It is used to access the device configuration registers as well as to transfer configuration data using the SelectMAP™ protocol.

The ICAP primitive (ICAP_VIRTEX2) provides access to device configuration data when connected to user logic in the FPGA fabric. It is only available in the Virtex-II and Virtex-II Pro families. The protocol used to communicate with ICAP is a subset of the SelectMAP protocol. ICAP, as indicated by its name, is an internally accessed resource and not intended for full device configuration. The PROGRAM, INIT, and DONE signals of SelectMAP are dedicated to initial or full device configuration and do not exist in the ICAP module (Figure 2, page 4). The ICAP module supports readback and partial reconfiguration of its own FPGA. This self-reconfiguration capability enables adaptive systems based on the PPC405 and the ICAP module. Due to the similarities of the ICAP and SelectMAP interface, any PPC405 designs based on ICAP can readily extend their reach to control MGTs within other Virtex-II Pro devices on the same board.

Consistent with the SelectMAP interface, the ICAP module provides read and write access to all configuration data. In the BERT reference design, the ICAP module is a slave IPIF on the PLB bus controlled by the PPC405 processor. More detailed information about the ICAP module can be found in the *Libraries Guide*.

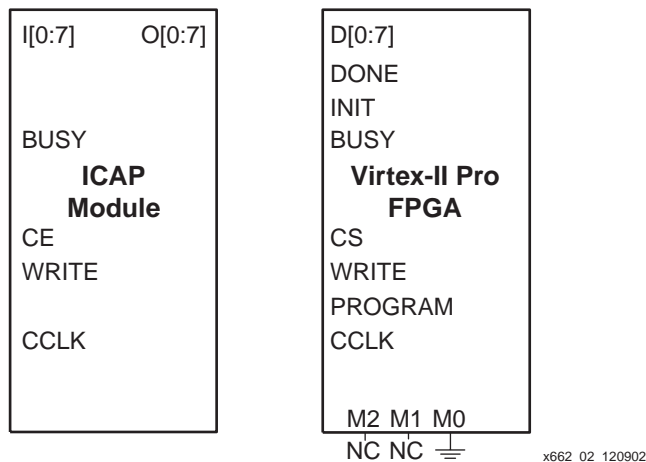


Figure 2: ICAP and SelectMAP Interfaces

ICAP IPIF and the IBM CoreConnect Bus

The BERT reference design that accompanies XAPP661 is PLB-based. XAPP662 also uses the PLB-based IPIF to ease integration to the BERT and minimize the FPGA resources required to incorporate the ICAP IPIF module. (See “Resource Utilization.”)

The PLB ICAP IPIF is implemented by connecting the ICAP module to a Xilinx PLB IPIF slave SRAM module (Figure 3, page 5). The PLB IPIF provides standardized PLB bus connections on one side and SRAM-like connections on the other side. The PLB IPIF makes it very easy to attach the ICAP module and associated control logic to the PLB CoreConnect bus while using very little glue logic. For more information regarding the Xilinx PLB IPIF slave SRAM module, see the Embedded Development Kit documentation. The PLB IPIF used in the PLB ICAP module provides a slave SRAM protocol as described in the IPIF specification. The version of the PLB IPIF used by the PLB ICAP only supports PLB non-burst memory transactions ($PLB_size[0:3] = 0000$, $PLB_type[0:2] = 000$) of one to four bytes. It ignores all other PLB transaction sizes and types.

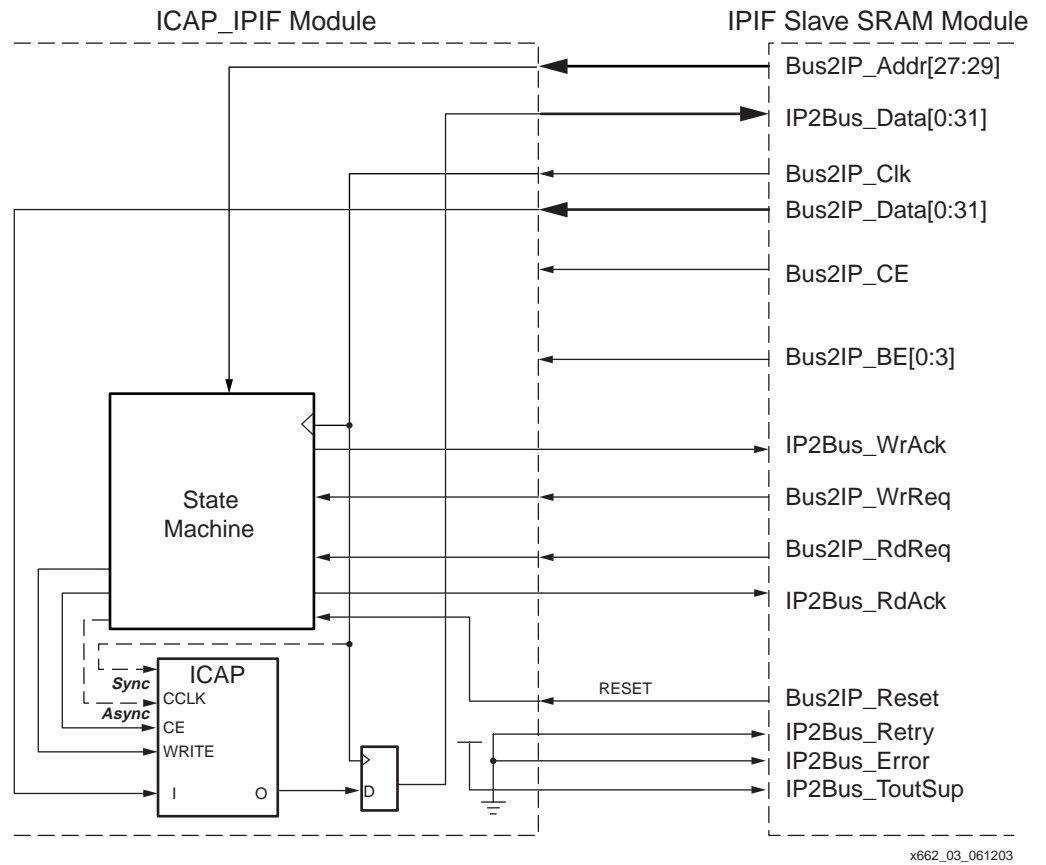


Figure 3: ICAP IPIF Interface to PLB Bus

ICAP IPIF

The ICAP IPIF, shown in [Figure 4](#), generates the appropriate ICAP module timing and control signals for the PLB BERT design implementation.

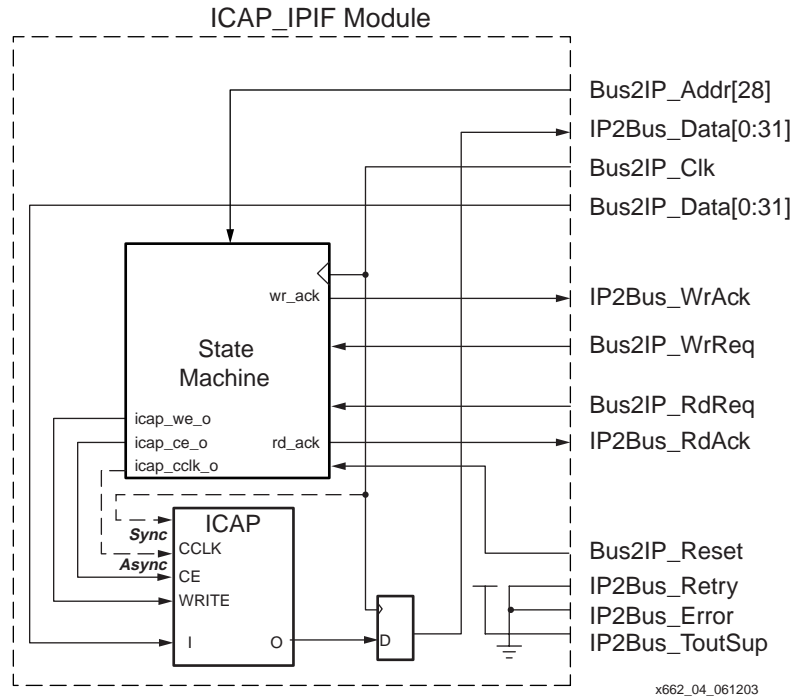


Figure 4: ICAP IPIF Control Logic Block Diagram

Consistent with the SelectMap interface and the clocking schemes described in [XAPP138](#), both the synchronous and asynchronous options of clocking are provided in this application note. The user can choose either the free-running CCLK (synchronous) or controlled CCLK (asynchronous) ICAP clocking when compiling the design.

To create a free-running CCLK design, define the constant "SYNCHRONOUS_ICAP" using the Verilog "define SYNCHRONOUS_ICAP" statement. To create an asynchronous controlled CCLK design simply leave out the constant definition statement. The free-running CCLK version wires the Bus2IP_Clk net to the ICAP CCLK clock port. The nets in and out if the ICAP module must be properly constrained in order to properly align the Data In, Data Out, and ICAP control signals with the ICAP CCLK clock.

Please refer to [XAPP138](#), "Virtex FPGA Series Configuration and Readback" for more details about the SelectMAP mode.

Free-Running CCLK ICAP IPIF

When the design is compiled using free-running CCLK ICAP clocking, the logic uses the Bus2IP_Clk as the ICAP module clock. The state machine in Figure 5 is used to generate the proper ICAP control signals when performing reads or writes to access configuration registers/data, or to abort the configuration process at any time. The Bus2IP_Clk for this reference design is actually an alias of the PLB clock.

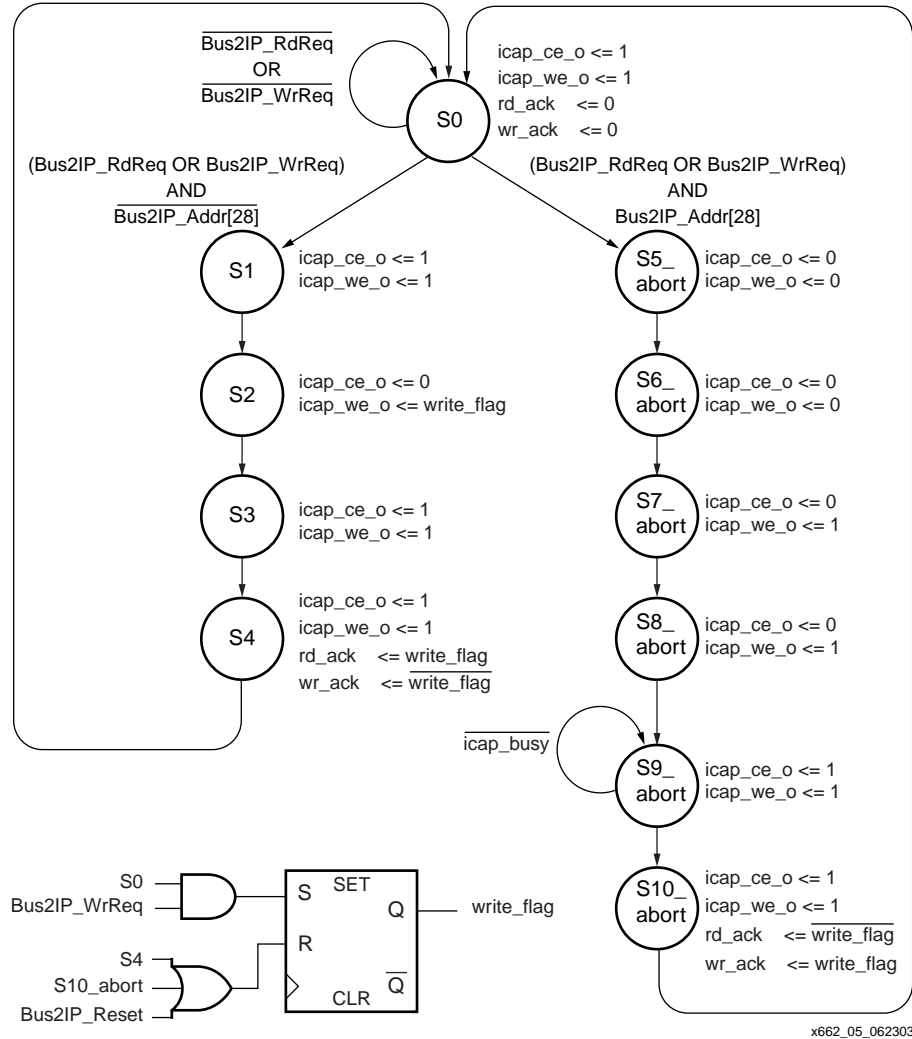


Figure 5: ICAP IPIF Control Logic with Free-Running ICAP Clocking

Controlled CCLK ICAP IPIF

When the design is compiled using controlled CCLK ICAP clocking, the logic uses a state machine generated clock as the ICAP module clock. The state machine shown in Figure 6 is used to generate the proper ICAP control signals when performing reads or writes to access configuration registers/data, or to abort the configuration process at any time. The ICAP module control pins are wired directly to the state machine outputs. Please refer to the reference code [XAPP661.zip](#) file for more detail.

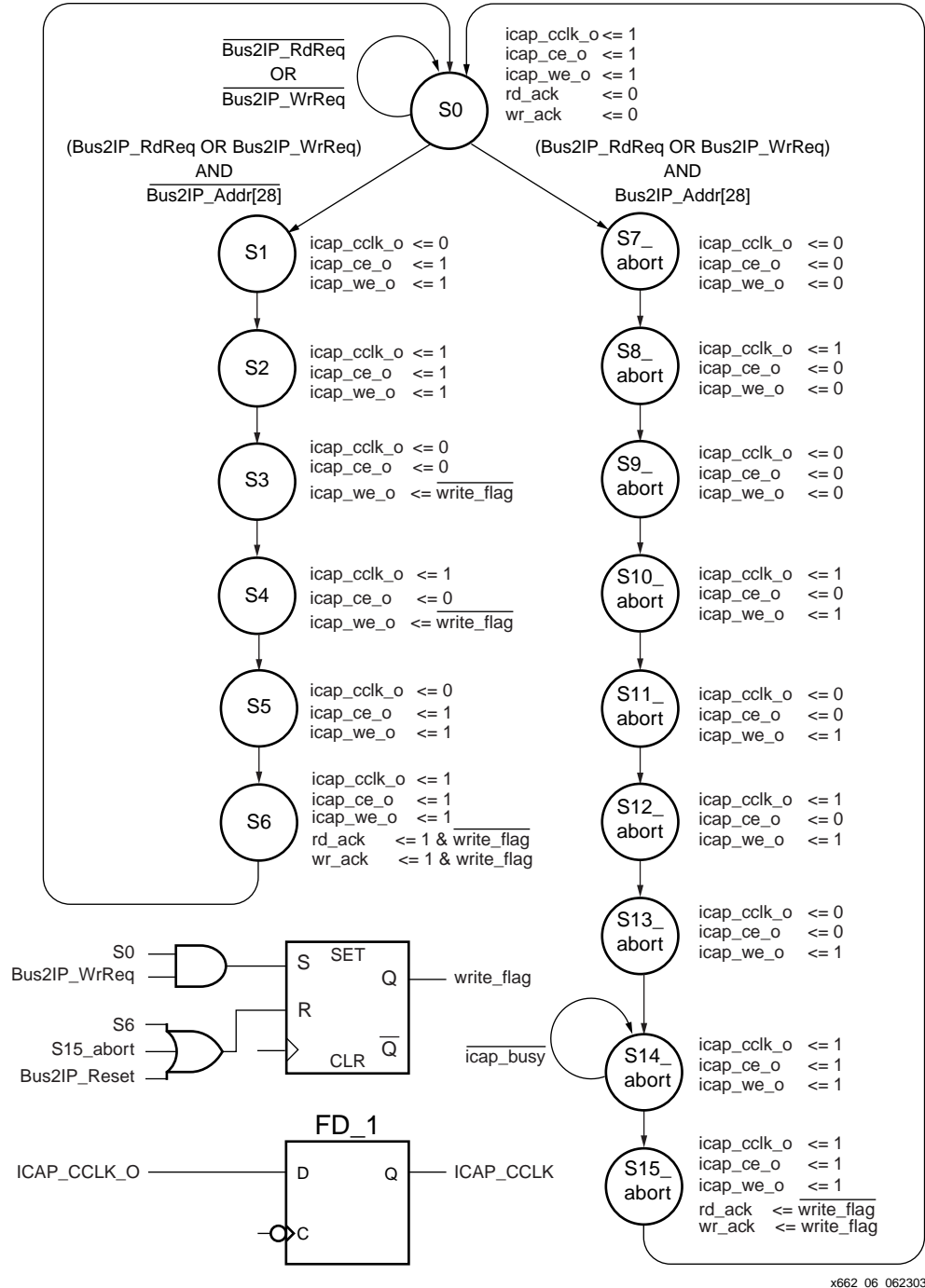


Figure 6: ICAP IPIF Control Logic with Controlled ICAP Clocking

Read Operation

The IPIF module compares the read address with the base address of the PLB ICAP module before it issues the read request to the ICAP module. The data output of the ICAP module consists of a single 8-bit word. Only the lower 8 bits [24:31] are wired to the output of the ICAP module, while bits [0:23] are not used and passed back to the IPIF as zeros.

Read-cycle timing for controlled CCLK operation is illustrated in [Figure 7](#), while free-running CCLK timing is illustrated in [Figure 8](#). These figures represent the previously described output of the state machines in the ICAP IPIF control logic. Please refer to *Controlled CCLK* and *Free Running CCLK* sections of [XAPP138](#) for compatible SelectMAP modes of operation.

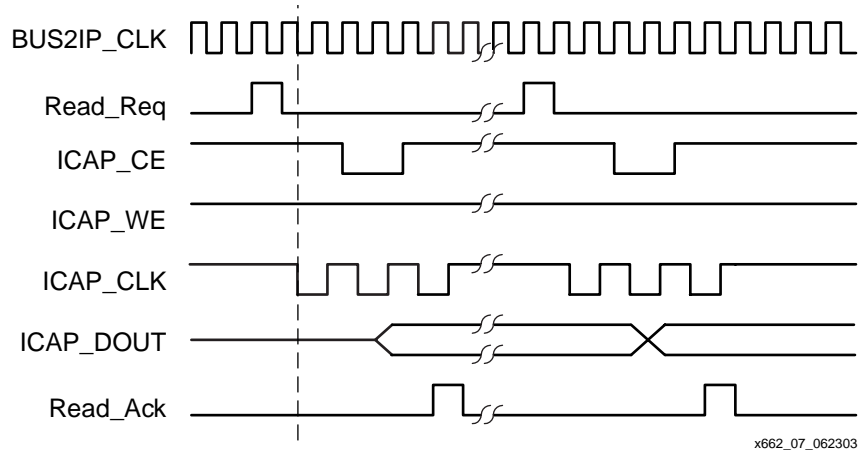


Figure 7: Read Cycle with Controlled ICAP Clocking

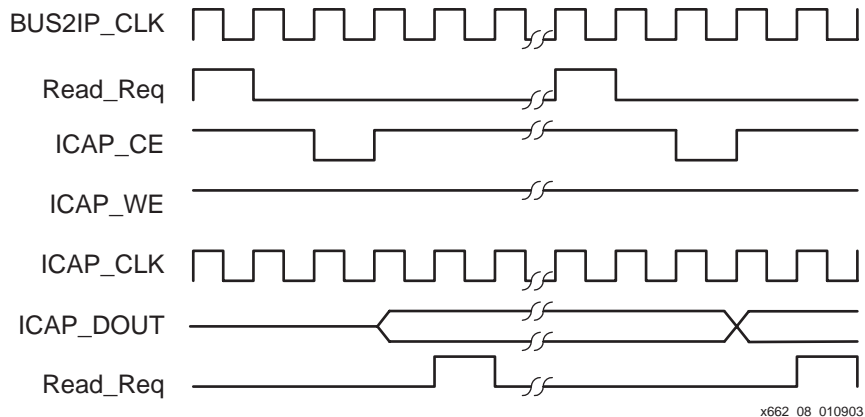


Figure 8: Read Cycle with Free-Running ICAP Clocking

Write Operation

The IPIF module compares the write address with the base address of the ICAP module before it issues the write request to the ICAP module. Since there is only a single 32-bit word to write to the ICAP, it is not necessary to multiplex on the Bus2IP_Addr in each write transaction. Only the lower 8 bits [24:31] are wired to the output of the ICAP module, while bits [0:23] are not used.

Write cycle timing for controlled CCLK operation is illustrated in [Figure 9](#) while free-running CCLK timing is illustrated in [Figure 10](#). The timing diagrams represent the output of the state machines in the previously described ICAP IPIF control logic. Please refer to *Controlled CCLK* and *Free Running CCLK* sections of [XAPP138](#) for compatible SelectMAP modes of operation.

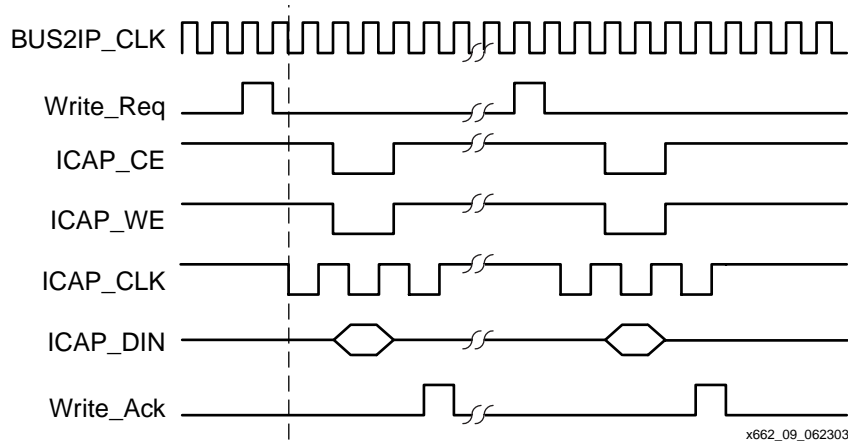


Figure 9: Write Cycle with Controlled ICAP Clocking

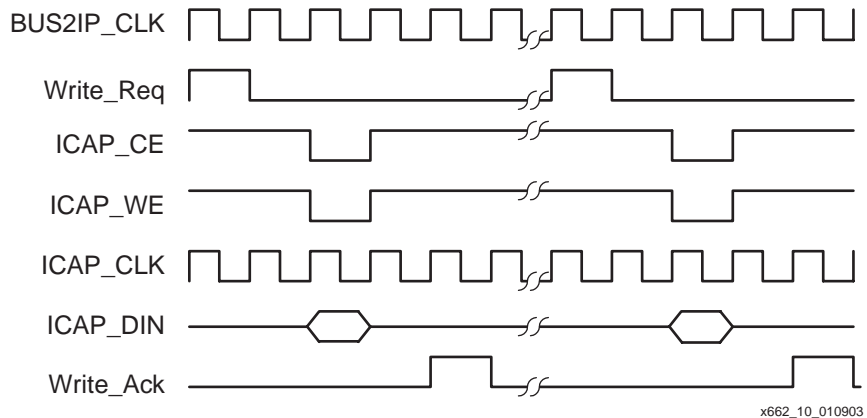


Figure 10: Write Cycle with Free-Running ICAP Clocking

Abort Operation

As described in the [Virtex-II Pro Platform FPGA User Guide](#), an ABORT is an interruption in the SelectMAP configuration or readback sequence which occurs when the state of WRITE_B changes while CS_B is asserted. The same is true for the ICAP block since it uses SelectMAP protocol.

After the ABORT sequence completes, the user must resynchronize the device. After resynchronizing, configuration can resume by sending the last configuration packet that was in progress when the ABORT occurred, or configuration can be restarted from the beginning.

The ABORT sequence can be generated at any time by performing a read or write access to the base address of the PLB ICAP +0x8 (Bus2IP_Addr[28] = 1).

Figure 11 shows the abort cycle timing for controlled ICAP clocking.

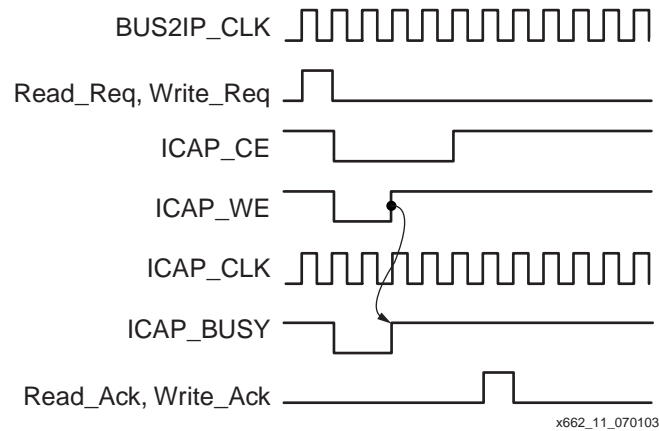


Figure 11: Read/Write Abort Cycle with Controlled ICAP Clocking

Figure 12 shows the abort cycle timing for free-running ICAP clocking.

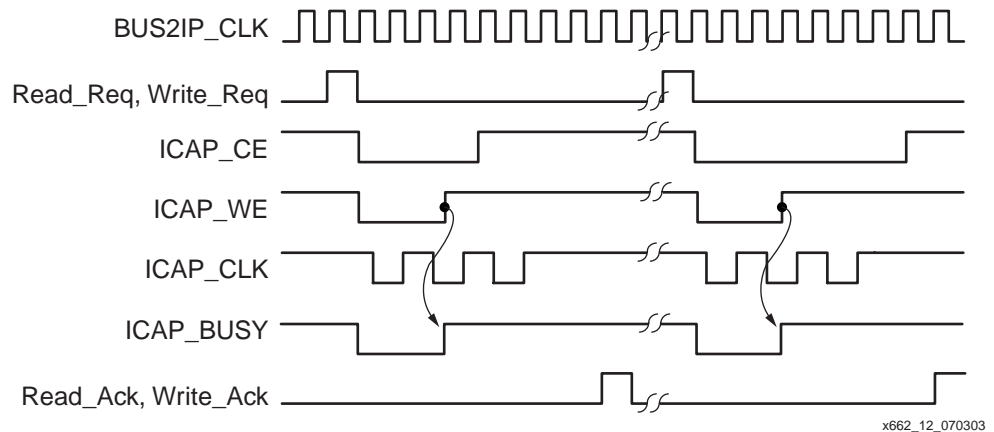


Figure 12: Read/Write Abort Cycle with Free-Running ICAP Clocking

Read/Write Acknowledgment

The PLB IPIF module supports read and write transactions on the CoreConnect bus. Bus masters on the PLB require proper read and write acknowledgement from the attached device to complete a transaction. The PLB ICAP IPIF uses a simple technique to generate the acknowledgement response once a request has been received. The ICAP IPIF module acknowledgement is generated after the corresponding read or write transaction completes at the ICAP module interface. Each slave device attached to the PLB is assigned a unique base address, distinguishing its read or write requests and acknowledgements from other slave devices.

Clock Domain

The ICAP IPIF is always synchronized to the Bus2IP_Clk. In this design, the Bus2IP_Clk is actually the PLB clock.

There are two ways to implement the ICAP module clocking, free-running and controlled CCLK. The free-running CCLK version wires the Bus2IP_Clk net to the ICAP clock port. The controlled CCLK version uses a state machine to generate the ICAP CCLK clock. The nets in and out of the ICAP module must be properly constrained in order to properly align the data in, data out, and ICAP control signals with the ICAP CCLK clock.

Other Devices on PLB

The PLB arbiter, PLB UART, and PLB BRAM controller are described in XAPP661, RocketIO transceiver BERT reference design. Please refer to [XAPP661](#) for more detailed information.

Resource Utilization

The ICAP IPIF provides a simple interface to the ICAP module for any design using the PPC405 and CoreConnect bus. The reference designs software code provides a powerful mechanism for updating MGT attributes. Initially, the reference design was developed for use with the Xilinx MGT characterization board suite, however, it is scalable to any Virtex-II Pro design using MGTs. An existing PLB based design with the ICAP IPIF is augmented by using relatively few FPGA fabric (logic and memory) resources because the bulk of the decision making control logic and the configuration data manipulation is performed by the PPC405 processor. Only the resources required to implement the ICAP IPIF and software memory storage are represented in [Table 1](#).

Table 1: FPGA Resource Utilization

ICAP IPIF Resources		Utilization by Device			
		XC2VP4	XC2VP7	XC2VP20	XC2VP50
LUTs Used	80	1.2%	0.7%	0.4%	0.2%
BRAMs Used	2 ⁽¹⁾	7.0%	4.6%	2.3%	0.9%

Notes:

1. For core RMW software functionality

Software Implementation

Overview

In designing a high-speed serial interconnect, getting data from point A to point B encompasses many engineering disciplines including semiconductor physics, device fabrication and packaging, board layout, material science, electromagnetics, and communication theory. Reliable communications requires the flexibility inherent in the Virtex-II Pro RocketIO transceivers. In particular, the ability to dynamically set the amount of pre-emphasis and the differential voltage swing allows a fielded system to compensate for the frequency characteristics of the path between two points. This compensation can be initiated by a user sitting in front of a terminal window through a human oriented user interface in the laboratory or it can be managed by the PPC405 hard core embedded in the Virtex-II Pro FPGA as a closed loop adaptive system. Whatever the form of the input stimulus, the underlying agent implementing the desired action is the read-modify-write (RMW) method of update. The core RMW functionality uses approximately two BRAMs (see [Table 1](#)), or the equivalent amount of storage in external RAM.

An adaptive system adjusts itself to its operating environment. There are two examples of operational awareness in the RocketIO RMW software. In the first example, the PPC405 uses ICAP to interrogate the FPGA for its IDCODE and for all enabled MGTs. This hardware scan allows the software to dynamically create user input menus where the MGT choices available for setting the RocketIO attributes reflects the currently instantiated MGTs.

The second example is the added capability provided by the user interface to select and update the MGT attributes and invoke the BERT in a lab setting. However, this process could be performed as an automated adaptive loop by the PPC405, for a fielded system ([Figure 13](#)). The same benefits of updating the MGT attributes can be applied in an adaptive fashion to any serial interface protocol.

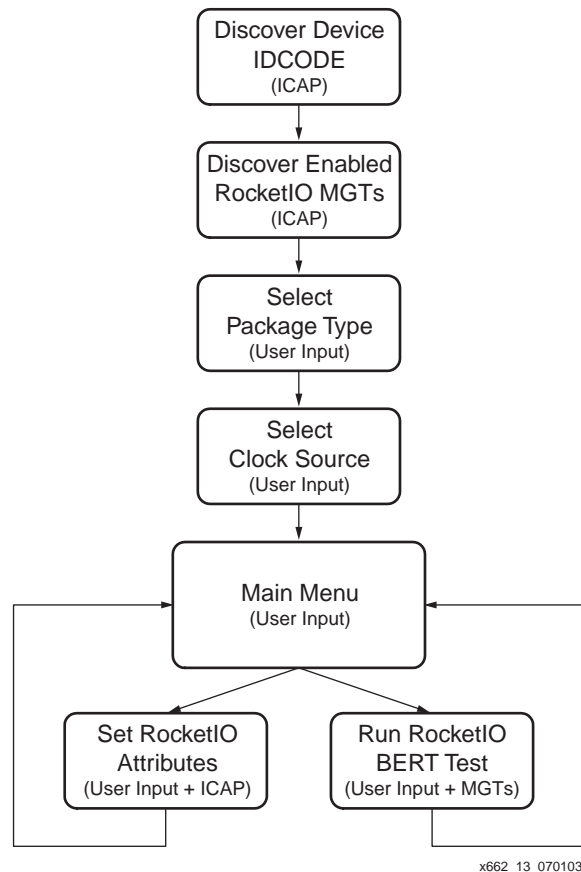


Figure 13: Typical Use Scenario

The RMW method of updating a RocketIO MGT frame involves reading a frame associated with an MGT, modifying the desired bits within the retrieved frame while leaving all the other frame bits unaltered, and writing back the updated MGT frame. This RMW flow uses the Virtex-II Pro ICAP block to configure the FPGA fabric. The software logic and data structures used to successfully update the RocketIO pre-emphasis, TX_PREEMPHASIS, and differential swing, TX_DIFF_CTRL, attributes of an MGT frame will be presented in this section. For further information on the bit-level details of the configuration bitstream please see the configuration section of the [Virtex-II Pro Platform FPGA User Guide](#).

The system discussed in this application note consists of a RocketIO characterization board (target) and a PC (host) connected to the target's serial port (19200, 8-N-1, No flow control). A simple command-line based user interface is provided through a host terminal window. A typical user flow through the terminal window menu screens is highlighted in the ["User Interface"](#) section. A detailed look at the RMW software implementation is provided in the ["RMW Software Implementation to Set RocketIO Attributes"](#) section.

User Interface

On power-up or after a full reconfiguration of the FPGA, a variety of setup operations are performed before the first menu is displayed on the terminal window. The software adapts to its environment by determining the IDCODE, saving the COR register state for later restoration, and scanning the underlying Virtex-II Pro FPGA for all enabled RocketIO transceivers. Direct software support is provided for IDCODEs matching the XC2VP4, XC2VP7, XC2VP20, XC2VP50, and XC2VP70 devices. After these initial steps, the RocketIO BERT Reference Design start-up screen is presented to the user. The start-up screen requires the user to consider the Virtex-II Pro package type and RocketIO transceiver clock source before entering the main menu. The device packages supported are FG456, FF672, FF1152 FF1517, and FF1704. The default setting is the FF672 package. The source clock for the RocketIO transceiver can be provided from an onboard crystal oscillator (BOARD OSC, default setting) or from an external signal generator connected to the board's SMA inputs (SMA setting). These two settings along with the automatic determination of the top or bottom location of the enabled MGTs provides the default clock setting passed on to the BERT characterization software when it is launched from the main menu.

[Figure 14](#) shows the menu selections available through both the start-up screen and the main menu. [Figures 15](#) through [22](#) are meant to take the user through a typical scenario from the initial Start-Up screen to setting the RocketIO attributes, running the BERT test, and querying some of the Virtex-II Pro registers. The sequence of screen shots and their captions is offered as a quick means to familiarize the reader with the software's user interface and capabilities before running any code.

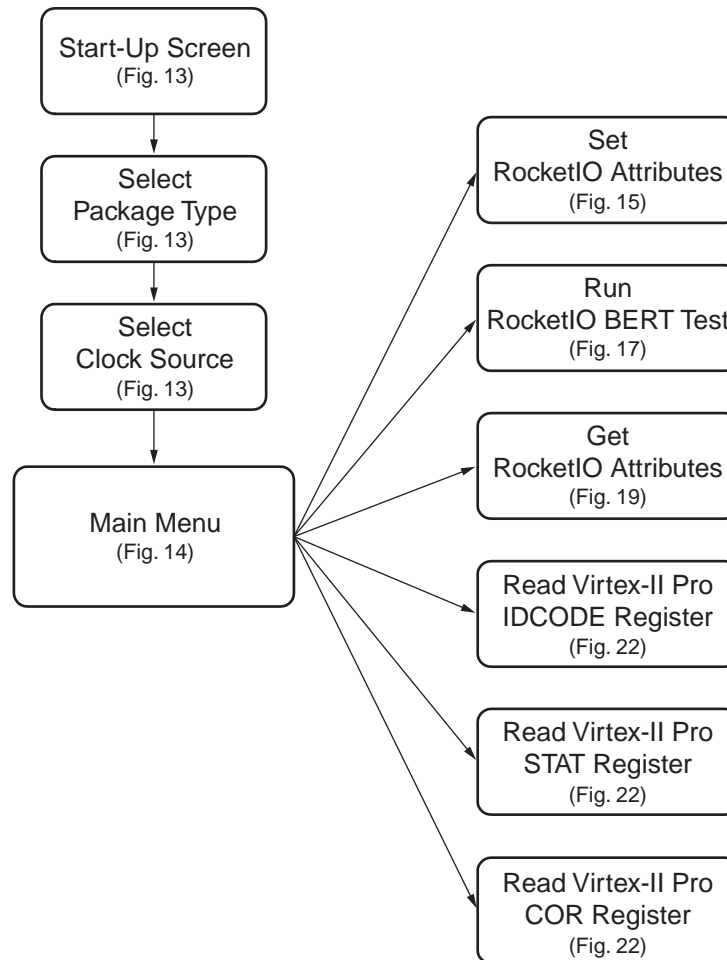
Navigating from the start-up screen ([Figure 15](#)) to the RocketIO BERT test ([Figure 19](#)) is accomplished by first picking selection #3 on the start-up screen and then selecting #2 from the main menu ([Figure 16](#)). Within a few seconds, the RocketIO transceiver status screen ([Figure 20](#)) becomes visible. If the RocketIO characterization board is physically connected for loopback operations, a line rate will be displayed. Please see [XAPP661](#) for details on the RocketIO BERT test.

Modifying the TX_PREEMPHASIS or the TX_DIFF_CTRL attributes of the RocketIO transceivers is done through selection #1. A user can switch back and forth, as desired, between selection #1, to calibrate the MGTs, and selection #2, to characterize the MGTs. Although selecting between calibration and characterization modes is offered to a user as a menu choice, this decision could be written as a PPC405 software control loop to adaptively optimize the RocketIO attributes.

[Figure 17](#) shows the MGT attribute update screen. The menu of available MGTs is dynamically created each time the update screen is entered. The names of the enabled MGTs and their physical locations on the Virtex-II Pro device is reflected by the row and column location of the MGT name in the menu. MGTs not enabled in the configured FPGA are denoted by asterisks.

Figure 21 shows the results of scanning the Virtex-II Pro device for all enabled RocketIO transceivers. The X,Y coordinates displayed are consistent with the coordinate system used by the Xilinx FPGA_EDITOR. For instance, a RocketIO MGT at X0,Y0 indicates the MGT on the bottom, left-hand side of the device, whereas an MGT at X0,Y1 is the top, left-hand side MGT. The four enabled MGTs shown in Figure 21 are on the bottom of the device.

In addition to the RocketIO calibration and characterization functionality, the main menu offers selections to read several of the configuration registers as shown in Figure 22. Bit level details of these registers are described in the Virtex-II Pro Platform FPGA user guide.



x662_14_070103

Figure 14: Menu Selection Flow

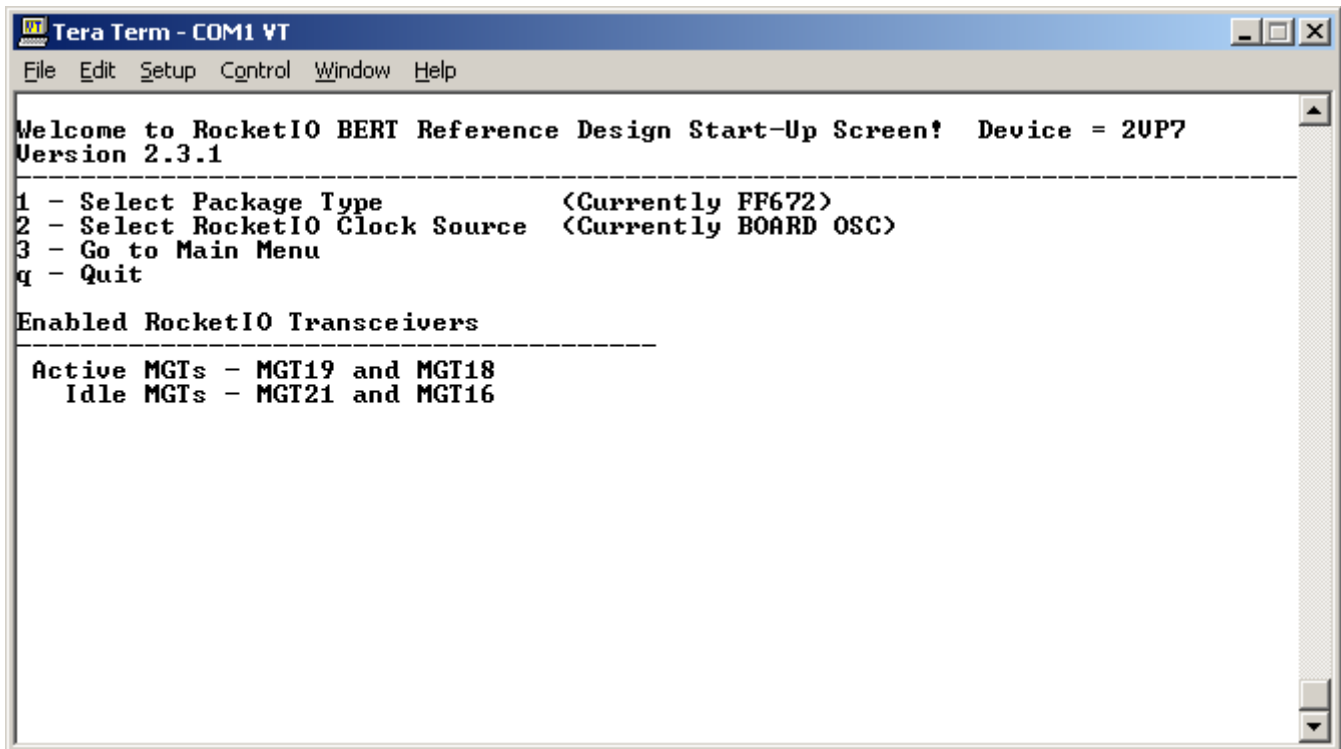


Figure 15: Start-up Screen. After Verifying the Package Type and Clock Source, Select #3 to Enter Main Menu

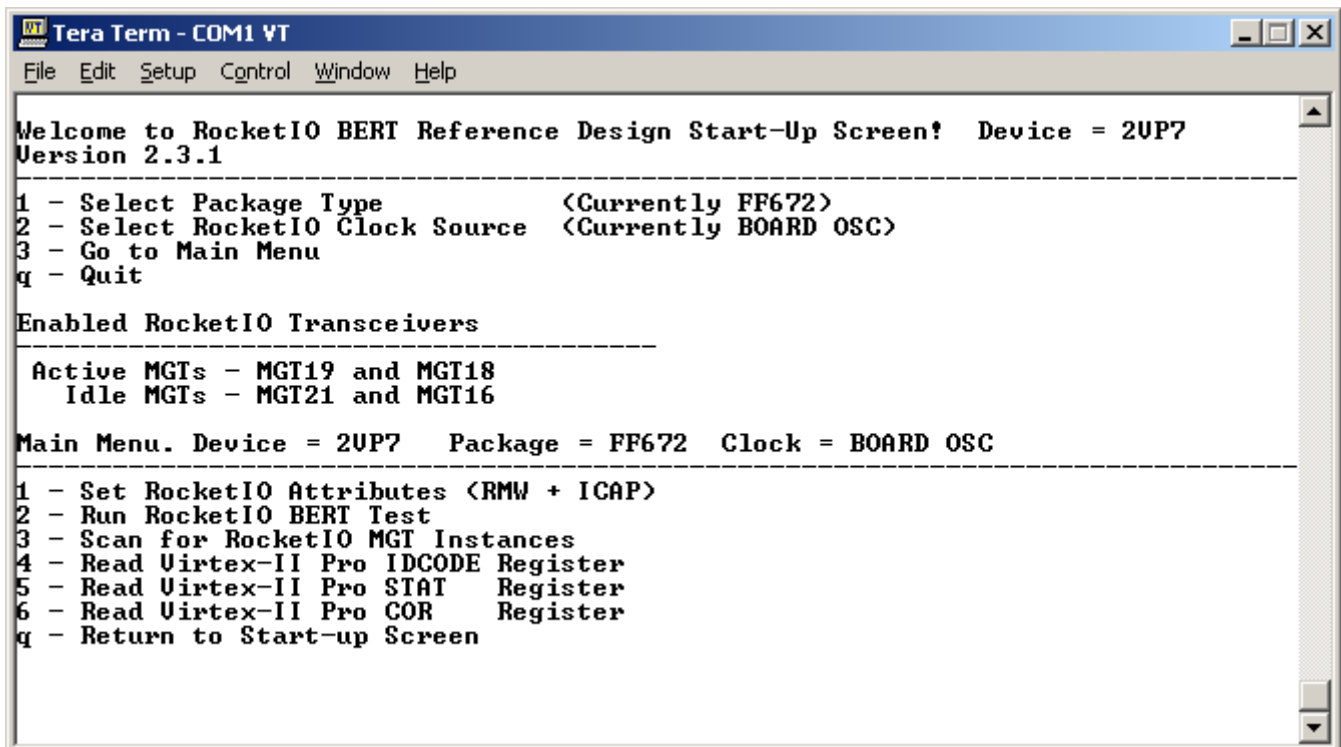


Figure 16: Main Menu: Select #2 to Run RocketIO BERT Test. Select #1 to Modify RocketIO Attributes

```

Tera Term - COM1 VT
File Edit Setup Control Window Help
Enabled RocketIO Transceivers
-----
Active MGIs - MGT19 and MGT18
Idle MGIs - MGT21 and MGT16
Main Menu. Device = 2UP7 Package = FF672 Clock = BOARD OSC
-----
1 - Set RocketIO Attributes <RMW + ICAP>
2 - Run RocketIO BERT Test
3 - Scan for RocketIO MGT Instances
4 - Read Virtex-II Pro IDCODE Register
5 - Read Virtex-II Pro STAT Register
6 - Read Virtex-II Pro COR Register
q - Return to Start-up Screen

RocketIO MGT Update. Device = 2UP7 Package = FF672 Clock = BOARD OSC
-----
Enter MGT number <1-4> where
Idle MGT 1 : MGT21
Idle MGT 2 : MGT16

< ***** ***** ***** ***** >
< 1 - MGT21 2 - MGT19 3 - MGT18 4 - MGT16 >
< q - Quit >

```

Figure 17: RocketIO MGT Selection During Attribute Update

```

Tera Term - COM1 VT
File Edit Setup Control Window Help
Enter MGT number <1-4> where
Idle MGT 1 : MGT21
Idle MGT 2 : MGT16

< ***** ***** ***** ***** >
< 1 - MGT21 2 - MGT19 3 - MGT18 4 - MGT16 >
< q - Quit >

TX_PREEMPHASIS Choices <Percent> are
0 - 10 1 - 20 2 - 25 3 - 33
Enter TX_PREEMPHASIS setting> 2

TX_DIFF_CTRL Choices are
0 - 400mV 1 - 500mV 2 - 600mV 3 - 700mV 4 - 800mV
Enter TX_DIFF_CTRL setting> 3

RocketIO MGT Update. Device = 2UP7 Package = FF672 Clock = BOARD OSC
-----
Enter MGT number <1-4> where
Idle MGT 1 : MGT21
Idle MGT 2 : MGT16

< ***** ***** ***** ***** >
< 1 - MGT21 2 - MGT19 3 - MGT18 4 - MGT16 >
< q - Quit >

```

Figure 18: RocketIO TX_PREEMPHASIS, TX_DIFF_CTRL Attribute Update

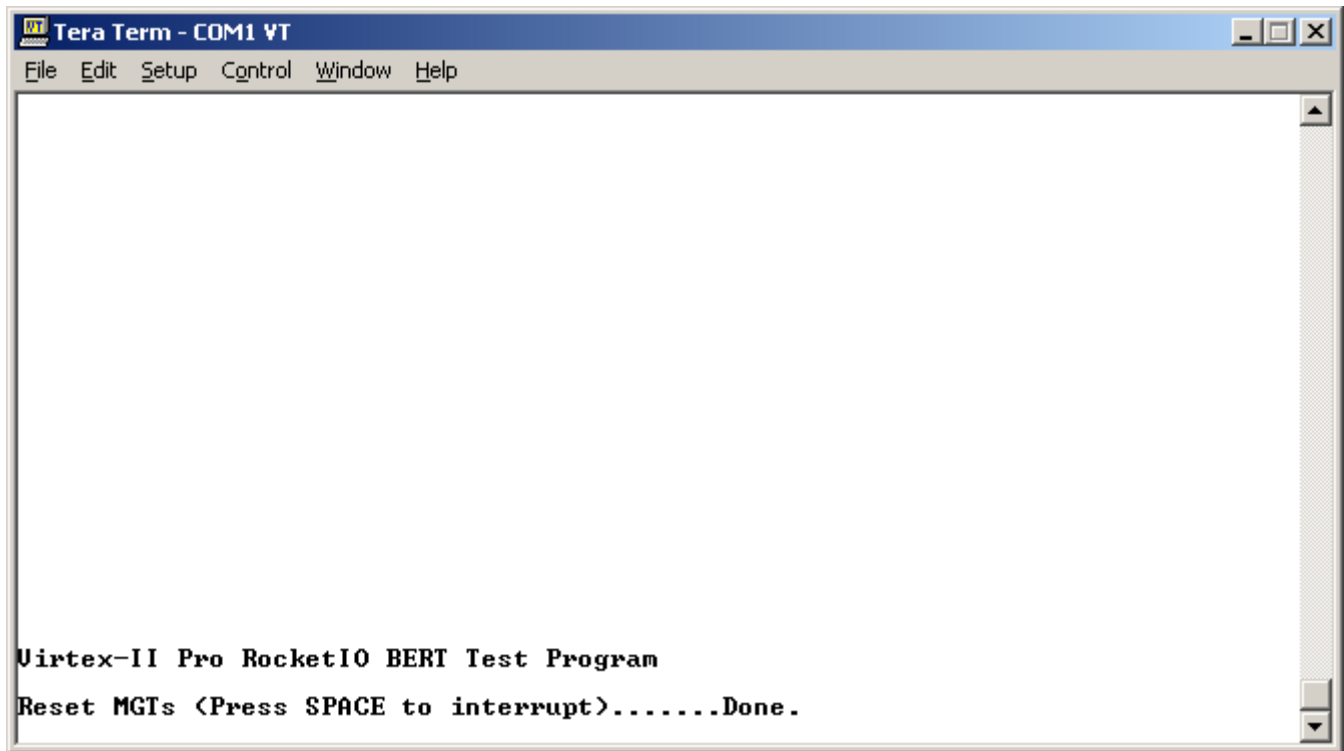


Figure 19: RocketIO BERT Test Launched

```

Tera Term - COM1 VT
File Edit Setup Control Window Help
-----
MGT# : RocketIO Transceiver ACTIVE MGT Status
-----
Received Frames # : 66,044 M + 847,131
Dropped Frames # : 0
Total Bit Errors # : 0
Bits Between Errs# : infinite
Line Rate # : 3,125 Mbps
MGT19: Bit Error Rate # : 0
Data Pattern # : 6-PRBS15: 1+x^14+x^15
Link-1 Abort-0 PowrDwn-0 TxInhibit-0 Loopbk-00

[1] Toggle TxInhibit [2] Toggle PowerDown
[3] Toggle Serial Loopback [7] Toggle ErrInsert
[d] Change Data Pattern
-----
Received Frames # : 66,044 M + 848,906
Dropped Frames # : 0
Total Bit Errors # : 0
Bits Between Errs# : infinite
Line Rate # : 3,125 Mbps
MGT18: Bit Error Rate # : 0
Data Pattern # : 7-PRBS20: 1+x^3+x^20
Link-1 Abort-0 PowrDwn-0 TxInhibit-0 Loopbk-00

[4] Toggle TxInhibit [5] Toggle PowerDown
[6] Toggle Serial Loopback [8] Toggle ErrInsert
[c] Change Data Pattern
-----
[+] Change Data Pattern on Both MGTs
[9] Switch MGT Clock Source <Current=BOARD OSC>
[0] Reset Both MGTs [ / ] Hide IDLE MGT menu
[ESC] Back to main menu
-----
MGT# : RocketIO Transceiver IDLE MGT Status
-----
MGT21: PowrDwn-0 TxInhibit-0 Loopbk-00

[I] Toggle TxInhibit [O] Toggle PowerDown
[P] Toggle Serial Loopback
-----
MGT16: PowrDwn-0 TxInhibit-0 Loopbk-00

[J] Toggle TxInhibit [K] Toggle PowerDown
[L] Toggle Serial Loopback
-----
Programmable Delay : 0
[>] Increase Delay [ < ] Decrease Delay
-----

```

Figure 20: RocketIO BERT Test Results: Updated Every Four Seconds. Press ESC to Return to Main Menu

```

Tera Term - COM1 VT
File Edit Setup Control Window Help
1 - Set RocketIO Attributes <RMW + ICAP>
2 - Run RocketIO BERT Test
3 - Scan for RocketIO MGT Instances
4 - Read Virtex-II Pro IDCODE Register
5 - Read Virtex-II Pro STAT Register
6 - Read Virtex-II Pro COR Register
q - Return to Start-up Screen

Device Type is 2UP7

Read Frame MGT_X = 0
           MGT_Y = 1
-----
Read Frame MGT_X = 1
           MGT_Y = 1
-----
Read Frame MGT_X = 2
           MGT_Y = 1
-----
Read Frame MGT_X = 3
           MGT_Y = 1
-----
Read Frame MGT_X = 0
           MGT_Y = 0
+++++ COMP ENABLED
      TX_PREEMPHASIS = 0 <10 Percent>
      TX_DIFF_CTRL = 400 mU
-----
Read Frame MGT_X = 1
           MGT_Y = 0
+++++ COMP ENABLED
      TX_PREEMPHASIS = 2 <25 Percent>
      TX_DIFF_CTRL = 700 mU
-----
Read Frame MGT_X = 2
           MGT_Y = 0
+++++ COMP ENABLED
      TX_PREEMPHASIS = 2 <25 Percent>
      TX_DIFF_CTRL = 700 mU
-----
Read Frame MGT_X = 3
           MGT_Y = 0
+++++ COMP ENABLED
      TX_PREEMPHASIS = 0 <10 Percent>
      TX_DIFF_CTRL = 500 mU
-----
Total Enabled MGTs Found = 4
    
```

**Figure 21: Scan for Enabled RocketIO MGTs. No Enabled MGTs Found on Top Portion (Y = 1) of XC2VP7
 Several MGTs Found with COMP Bit Enabled on Bottom Portion (Y = 0) of XC2VP7
 Total of Four Enabled MGTs Found in Current FPGA Configuration**

```

Tera Term - COM1 VT
File Edit Setup Control Window Help
Main Menu. Device = 2UP7 Package = FF672 Clock = BOARD OSC
-----
1 - Set RocketIO Attributes <RMW + ICAP>
2 - Run RocketIO BERT Test
3 - Scan for RocketIO MGT Instances
4 - Read Virtex-II Pro IDCODE Register
5 - Read Virtex-II Pro STAT Register
6 - Read Virtex-II Pro COR Register
q - Return to Start-up Screen

IDCODE: 0x11 0x24 0xA0 0x93

Main Menu. Device = 2UP7 Package = FF672 Clock = BOARD OSC
-----
1 - Set RocketIO Attributes <RMW + ICAP>
2 - Run RocketIO BERT Test
3 - Scan for RocketIO MGT Instances
4 - Read Virtex-II Pro IDCODE Register
5 - Read Virtex-II Pro STAT Register
6 - Read Virtex-II Pro COR Register
q - Return to Start-up Screen

STATUS REG: 0x00 0x00 0x1F 0xEC

Main Menu. Device = 2UP7 Package = FF672 Clock = BOARD OSC
-----
1 - Set RocketIO Attributes <RMW + ICAP>
2 - Run RocketIO BERT Test
3 - Scan for RocketIO MGT Instances
4 - Read Virtex-II Pro IDCODE Register
5 - Read Virtex-II Pro STAT Register
6 - Read Virtex-II Pro COR Register
q - Return to Start-up Screen

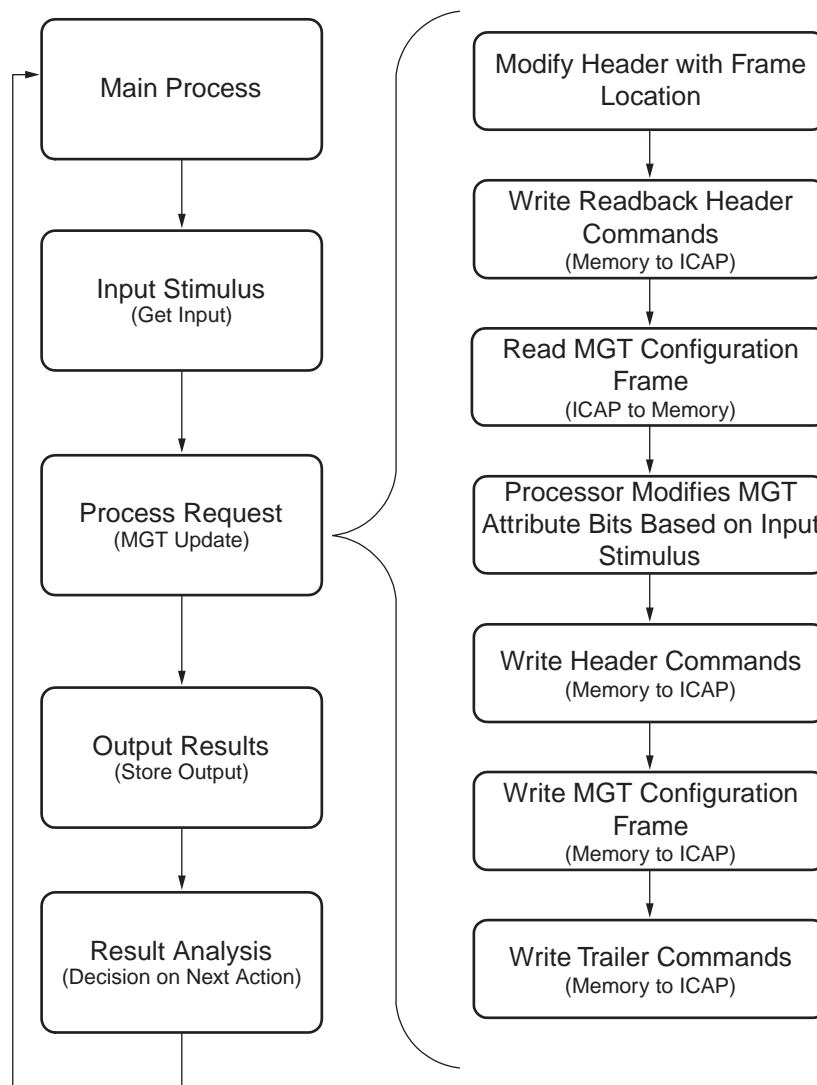
COR: 0x25 0x05 0x3F 0xF6

```

Figure 22: Results of Reading Virtex-II Pro IDCODE, STAT, and COR Registers

RMW Software Implementation to Set RocketIO Attributes

Virtex-II Pro devices are configured or queried by writing and reading a set of internal configuration registers. The internal configuration memory is partitioned into segments called “Frames.” Though the number and size of the frames varies with the device size, access to configuration data is quantized at the frame level. Each RocketIO transceiver is associated with its own configuration frame. There are many RocketIO transceiver attributes that can be set through the configuration bitstream loaded at power-up. The configuration bitstream, a sequence of configuration commands and data frames, sets the initial value of the RocketIO attributes. By using the RMW software, changes to the transceiver attributes can be made after the initial configuration operation, without re-configuring the entire FPGA. A generalized view of the process of assembling the header commands, the data frame, and trailer commands to affect a RocketIO attribute change is shown in Figure 23.



x662_25_070103

Figure 23: RMW Flow to Update MGT Frames

The differential voltage swing, TX_DIFF_CTRL, of a RocketIO transceiver can take on five settings. The transceiver pre-emphasis, TX_PREEMPHASIS, can be set from 10% to 33% through four available presets. Table 2 and Table 3 lists the RocketIO attributes modifiable by the reference design software.

Table 2: Differential Voltage Swing Settings

Setting Number	FPGA_EDITOR Setting	TX_DIFF_CTRL Attribute	C-Code Attribute Byte Value
1	400	400 mV	0x20
2	500	500 mV	0x00
3	600	600 mV	0x40
4	700	700 mV	0x60
5	800	800 mV	0x22

Table 3: Transceiver Pre-emphasis Settings

Setting Number	FPGA_EDITOR Setting	TX_PREEMPHASIS Attribute	C-Code Attribute Byte Value
1	0	10%	0x00
2	1	20%	0x10
3	2	25%	0x08
4	3	33%	0x18

Header

When writing, reading a configuration frame, or accessing the Virtex-II Pro internal configuration registers, the RMW software uses a set of pre-defined static C-code arrays to initiate the conversation with the Virtex-II Pro processor's configuration circuitry and send the appropriate command sequence. Each header section begins with a dummy word (0xFFFFFFFF) followed by the sync word (0xAA995566). The FPGA configuration circuitry uses the sync word to establish 32-bit alignment of the incoming byte stream. Table 4 shows the types of pre-defined headers that are currently used by the RMW software.

Table 4: Pre-defined Command Headers

Read STAT Register
Read IDCODE Register
Write Configuration Frame
Read Configuration Frame
Read COR Register
Write COR Register

Operations that read configuration registers consist of a *header* section followed by a *trailer* section without the need for an ID code or frame address information. Each configuration register's address is embedded within the command. However, when writing a configuration register the device IDCODE is required to be included in the command sequence. Similarly, reading or writing frame data requires the device IDCODE, and additionally the address of the targeted frame. For example, the write configuration frame and read configuration frame headers contain a device specific product ID code (Table 5). This code is automatically set by the software to reflect the FPGA the software is executed on. The byte offset into the Write (WCFG) and Read (RCFG) frame command header is also listed in Table 5. An X, Y coordinate system consistent with the view provided by FPGA_EDITOR is used to address frames within Virtex-II Pro devices (Table 6 and Table 7). The hexadecimal values

corresponding to the X and Y frame addresses for each MGT are also listed. Within this document, the static header arrays shown are for an XC2VP7 device.

Table 5: Command Header IDCODE and Offset

FPGA	IDCODE (HEX)	Address Byte Offset (RCFG)		Address Byte Offset (WCFG)	
		X	Y	X	Y
		XC2VP4	v1 23 E0 93	29	30
XC2VP7	v1 24 A0 93	29	30	37	38

Notes:

1. The letter "v" indicates a device revision code in HEX

Table 6: Bottom MGT Addressing and Naming

X	Y	XC2VP4	XC2VP7	X Address Byte Value	Y Address Byte Value
0	0	MGT19	MGT21	0x02	0x06
1	0	MG18	MGT19	0x04	0x06
2	0		MGT18	0x06	0x06
3	0		MGT16	0x08	0x06

Table 7: Top MGT Addressing and Naming

X	Y	XC2VP4	XC2VP7	X Address Byte Value	Y Address Byte Value
0	1	MGT6	MGT4	0x02	0x00
1	1	MG17	MGT6	0x04	0x00
2	1		MGT7	0x06	0x00
3	1		MGT9	0x08	0x00

Read Status Register

The header sequence of 20 bytes is written byte by byte to the ICAP port beginning with array index 0 to array index 19. Once the header is written, a dummy byte read is performed followed by a read of the expected four status bytes. Communication with the ICAP port terminates with the sending of the *trailer* sequence.

```
static char read_status[20] = {
    0xFF, 0xFF, 0xFF, 0xFF, /* Dummy Word */
    0xAA, 0x99, 0x55, 0x66, /* Sync Word */
    0x28, 0x00, 0xE0, 0x01, /* Read Status Register */
    0x00, 0x00, 0x00, 0x00, /* Pad */
    0x00, 0x00, 0x00, 0x00, /* Pad */
};
```

Read ID Code Register

The header sequence of 20 bytes is written byte by byte to the ICAP port beginning with array index 0 to array index 19. Once the header is written, a dummy byte read is performed, followed by a 4-byte read of the product ID code. Communication with the ICAP port terminates with the sending of the *trailer* sequence.

```
static char read_id[20] = {
    0xFF, 0xFF, 0xFF, 0xFF, /* Dummy Word */
    0xAA, 0x99, 0x55, 0x66, /* Sync Word */
    0x28, 0x01, 0xC0, 0x01, /* Read Product ID Code Register */
    0x00, 0x00, 0x00, 0x00, /* Pad */
    0x00, 0x00, 0x00, 0x00 /* Pad */
};
```

Write Configuration Frame

The header sequence of 44 bytes is written byte by byte to the ICAP port beginning with array index 0 to array index 43. Once the header is written, the actual frame data is written, followed by a pad frame. Communication with the ICAP port terminates with the sending of the *trailer* sequence.

```
static char cmd_writeframe[44] = {
    0xFF, 0xFF, 0xFF, 0xFF, /* Dummy Word */
    0xAA, 0x99, 0x55, 0x66, /* Sync Word */
    0x30, 0x00, 0x80, 0x01, /* Write to CMD Register */
    0x00, 0x00, 0x00, 0x07, /* Command RCRC - Reset CRC Register */
    0x30, 0x01, 0xC0, 0x01, /* Write ID Code Register with next word */
    0x01, 0x24, 0xA0, 0x93, /* XC2VP7 Product Code written to ID Register */
    0x30, 0x00, 0x80, 0x01, /* Write to CMD Register */
    0x00, 0x00, 0x00, 0x01, /* Command WCFG - Write Configuration Data */
    0x30, 0x00, 0x20, 0x01, /* Write to Frame Address Register(FAR) */
    0x04, 0x02, 0x00, 0x00, /* MGT Frame Address of Interest */
    0x30, 0x00, 0x40, 0xD4, /* Write 212 words to Frame Data Input
                                Register(FDRI) */
};
```

Read Configuration Frame

The header sequence of 44 bytes is written byte by byte to the ICAP port beginning with array index 0 to array index 43. Once the header is written, a dummy byte read is performed, followed by a pad frame read and then the read back of the actual frame data. Communication with the ICAP port terminates with the sending of the *trailer* sequence.

```
static char cmd_readframe[44] = {
    0xFF, 0xFF, 0xFF, 0xFF, /* Dummy Word */
    0xAA, 0x99, 0x55, 0x66, /* Sync Word */
    0x30, 0x01, 0xC0, 0x01, /* Write ID Code Register with next word */
    0x01, 0x24, 0xA0, 0x93, /* XC2VP7 Product Code written to ID Register */
    0x30, 0x00, 0x80, 0x01, /* Write to CMD Register */
    0x00, 0x00, 0x00, 0x04, /* Command RCFG - Read Configuration Data */
    0x30, 0x00, 0x20, 0x01, /* Write to Frame Address Register(FAR) */
    0x04, 0x02, 0x00, 0x00, /* MGT Frame Address of Interest */
    0x28, 0x00, 0x60, 0x00, /* Read from Frame Data Output Register(FDRO) */
    0x48, 0x00, 0x00, 0xD4, /* Read 212 words in command given above */
    0x00, 0x00, 0x00, 0x00 /* Pad */
};
```

Read/Write Configuration Options Register

The configuration options register (COR) is used in the RMW flow to disable CRC checking during the RocketIO MGT frame update. Upon entry to the RMW software the CRC_BYPASS bit in the COR is set to force all CRC checks to match against the predetermined value of 0x0000DEFB. The COR stores many bitgen created user options so a RMW of the COR

register is done when altering CRC_BYPASS. Before exiting the RMW software, the COR is restored to its original contents. The header section of the command used to read the COR, and the header section for the COR write operation follow. The contents written to the COR are the values present in the *write_cor[]* command array at byte offset 20 to byte offset 23. The values shown are example values.

Read Configuration Options Register

```
static char read_cor[20] =
{0xFF, 0xFF, 0xFF, 0xFF, /* Dummy Word */
0xAA, 0x99, 0x55, 0x66, /* Sync Word */
0x28, 0x01, 0x20, 0x01, /* Read Configuration Options Register */
0x00, 0x00, 0x00, 0x00, /* Pad */
0x00, 0x00, 0x00, 0x00 /* Pad */
};
```

Write Configuration Options Register

```
static char write_cor[24] = {
0xFF, 0xFF, 0xFF, 0xFF, /* Dummy Word */
0xAA, 0x99, 0x55, 0x66, /* Sync Word */
0x30, 0x01, 0xC0, 0x01, /* Write ID Code Register with next word */
0x01, 0x24, 0xA0, 0x93, /* XC2VP7 Product Code written to ID Register */
0x30, 0x01, 0x20, 0x01, /* Write COR with next word */
0x01, 0x05, 0x3F, 0xE5 /* Used to set/reset CRC Bypass in COR */
};
```

Configuration Frame

Once the desired RocketIO attribute settings are entered by the user through the terminal interface, the addressed MGT frame is read via the ICAP port and placed in a frame memory buffer. A device dependent sized frame buffer holds a single MGT frame. The RMW software allocates 424 bytes for the frame buffer, covering both the XC2VP4 and XC2VP7 devices. Once read into the frame buffer, only the MGT frame TX_PREEMPHASIS and TX_DIFF_CTRL attributes are updated and then the frame is written back out the ICAP port to dynamically re-configure the addressed MGT. There are a variety of C-code constants defined for the four possible pre-emphasis settings and the five possible differential swing settings. Table 2 and Table 3 show the byte values associated with the TX_PREEMPHASIS and TX_DIFF_CTRL attributes. An attribute offset from the beginning of the frame is also defined for both top and bottom MGTs. The attribute location in the frame differs depending on the top or bottom location of the RocketIO transceiver (Table 8 and Figure 24) .

Table 8: Attribute Byte Offset and Frame Sizes

Device	Number of Frames	Frame Length (Bytes)	Attribute Byte Offset (Top MGT)	Attribute Byte Offset (Bottom MGT)
XC2VP4	884	424	2	412
XC2VP7	1,320	424	2	412

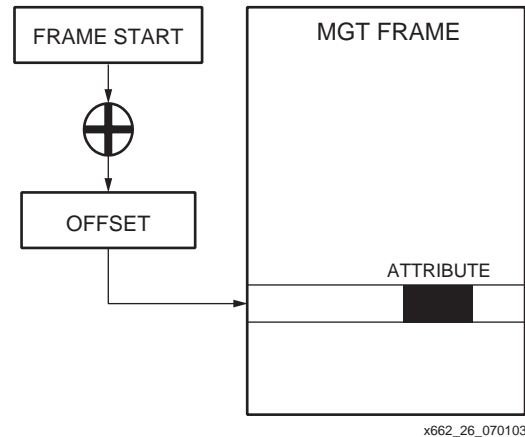


Figure 24: Attribute Location Based on Table 8 Offset Values

Trailer

Terminating each command sequence, a DESYNCH command (Table 9) is written to the Virtex-II Pro device forcing a subsequent re-synchronization with a SYNC word to re-establish 32-bit boundaries. This allows code (independent of the state of the 32-bit alignment) to be written. Alignment is not guaranteed and each command sequence is independently responsible for establishing 32-bit alignment.

Table 9: Command Trailer

0x30, 0x00, 0x80, 0x01	/* Write CMD Register */
0x00, 0x00, 0x00, 0x0D	/* Command – DESYNCH */

RMW Frame Logic

The RMW software contains two key routines to combine the *header*, configuration frame, and *trailer* functions; *readframe()* and *writeframe()*. The logic flow in the *readframe()* routine is listed in Table 10, while the logic for *writeframe()* is listed in Table 11. Once an MGT frame is retrieved and stored in memory, its attributes can be modified and written back to the addressed MGT.

Table 10: Steps to Read a Frame

Step Number	Description
1	Write cmd_readframe[] array with desired MGT address
2	Perform a dummy byte read
3	Read a pad frame (424 Bytes for XC2VP7)
4	Read and store actual MGT data frame (424 Bytes for XC2VP7)
5	Write DESYNCH command

Table 11: Steps to Write a Frame

Step Number	Description
1	Write cmd_writeframe[] array with desired MGT address
2	Write stored and modified frame buffer (424 Bytes for XC2VP7)
3	Write a pad frame (424 Bytes for XC2VP7)
4	Write CRC (0x0000DEFC) when CRC disabled option selected
5	Write DESYNCH command
6	Write four dummy no-OP words (0x20000000)

RocketIO BERT Test Software Interface

The main menu interface to the RocketIO BERT test is through a call to the *bert_main()* function. Once the BERT test is completed, the user can press the ESCAPE key to return to the main menu. The global variable BREFCLK_SEL is set in the start-up screen software before *bert_main()* can be called from the main menu. The value BREFCLK_SEL takes depends on three factors, the package type, the clock source, and the location (either top or bottom) of the RocketIO transceivers instantiated in BERT hardware IP. Table 12 and Table 13 show how the value BREFCLK_SEL is assigned to either BREFCLK (0) or BREFCLK 2 (1).

Table 12: BREFCLK_SEL Setting for Top MGTs

Package Type	Onboard Oscillator	SMA Connector
FG456	BREFCLK	BREFCLK 2
FF672	BREFCLK 2	BREFCLK
FF1152	BREFCLK 2	BREFCLK

Table 13: BREFCLK_SEL Setting for Bottom MGTs

Package Type	Onboard Oscillator	SMA Connector
FG456	BREFCLK 2	BREFCLK
FF672	BREFCLK	BREFCLK 2
FF1152	BREFCLK	BREFCLK 2

Additional Utility Functions

There are times when reading the Virtex-II Pro configuration registers and MGT frames can aid in debugging. In keeping with this spirit, there are several menu selections allowing a query of some of the configuration registers. Figure 22 shows the results of querying the IDCODE, STAT, and COR registers within the Virtex-II Pro device. These utilities are accessed from main menu selections #4 through #6, and invoke the *get_idcode()*, *get_status()*, and *set_cor()* functions respectively. The command sequences used to implement these register operations are shown in the “Read Status Register”, “Read ID Code Register”, “Read Configuration Options Register”, and “Write Configuration Options Register” sections.

Discovering the location of enabled MGTs and their TX_PREEMPHASIS and TX_DIFF_CTRL attribute settings is also a helpful function. Selection #3 from the Main Menu, “3 - Scan for MGT Instances,” uses the RMW flow to search the device for all enabled RocketIO transceivers and display their location and controllable attribute settings. Figure 21 shows the output resulting from scanning for MGTs. The MGT search sequentially looks for MGT frames with their COMP attribute bit set and notes this in a two row by ten column static array named

mgt_enabled[2][10]. The rows of this array correspond to the top or bottom location of the MGTs (Y coordinate), while the columns represent the X coordinate. Each enabled RocketIO transceiver is also added to an array of MGT_ENTRY structures (enabled_mgts[20]). This is used to dynamically create the names and identify the location of all enabled MGTs whenever the RocketIO attribute update menu is displayed.

Conclusion

This application note describes the in-circuit partial reconfiguration of RocketIO transceiver attributes using the Virtex-II Pro internal configuration access port (ICAP). This solution uses a Virtex-II Pro device with an IBM PowerPC 405 (PPC405) processor to perform a partial reconfiguration of the RocketIO multi-gigabit transceiver's (MGTs) pre-emphasis and differential swing control attributes. As a specific example of creating a system with an ICAP module and a custom hardware module, the RocketIO transceiver BERT reference design is used as a basis for this application note. A common reference design is provided to cover XAPP661 and XAPP662. The C language design presents an adaptive software solution, applicable to any serial interface protocol, for upgrading MGTs using the PPC405 and ICAP. The XAPP662 design is bundled with the XAPP661 reference design and can be downloaded from the Xilinx web site at: <http://www.xilinx.com/bvdocs/appnotes/xapp661.zip>.

Reference

1. Xilinx, Inc., [Embedded Development Kit](http://www.xilinx.com/edk). (<http://www.xilinx.com/edk>)
2. Xilinx, Inc., [XAPP138: Virtex Configuration and Readback](#).
3. Xilinx, Inc., [XAPP660: Partial Reconfiguration of RocketIO Pre-emphasis and Differential Swing Control Attributes](#) by Derek R. Curd.
4. Xilinx, Inc., [XAPP661: RocketIO Transceiver Bit-Error Rate Tester](#) by Dai Huang and Mike Matera.
5. Xilinx, Inc., [RocketIO Transceiver User Guide](#).
6. Xilinx, Inc., [Virtex-II Pro Platform FPGA User Guide](#).

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/13/03	1.0	Initial Xilinx release.
07/03/03	1.1	Added "Abort Operation" section and support for EDK.
05/26/04	2.4	New screenshots and version number to match Version 2.4 of xapp661.zip. See readme.txt in zip file for version number. Updated URLs.